Dokumentacja końcowa

Modyfikacja XLNet dla określania podobieństwa semantycznego (iSTS) dwóch zdań w języku angielskim

Tomasz W., Artur G.

01.06.2021

Spis treści

1. Cel projektu

Celem projektu jest modyfikacja XLNet dla określania podobieństwa semantycznego (iSTS) dwóch zdań w Jeżyku angielskim oraz odniesienie się od wyników uzyskanej w pracy magisterskiej Pani Eweliny Grudzień[1]. Na pracę prowadzone w ramach tego projektu składało się zapoznanie się z architekturą rozwiązania XLNet oraz artykułem opisującym to rozwiązanie, modyfikacja rozwiązania implementującego XLNet na potrzeby zadania iSTS oraz porównanie otrzymanych wyników do wyników opisanych w pracy magisterskiej na podstawie zbiorów danych udostępnionych w ramach konkursu SemEval w latach 2015 i 2016.

2. Charakterystyka zadania iSTS

Zadanie iSTS (ang. interpretable semantic textual similarity) ma na celu określenie stopnia podobieństwa semantycznego pomiędzy parami fragmentów tekstu, a także wyjaśnienie przyznanej oceny poprzez określenie typu relacji (lub braku relacji) pomiędzy nimi. Zadanie to jest swoistym rozszerzeniem zadani podobieństwa semantycznego (STS), które ogranicza się do wyznaczenia tylko oceny podobieństwa a dodanie ów typu relacji/dopasowania nakłada dodatkową warstwę interpretowalności.

2.1. Ocena podobieństwa

Ocena podobieństwa opisuję podobieństwo i relację między badanymi fragmentami tekstu (ang. *chunks*) w skali od 0 do 5, co sprowadza się do 6 kategorii w postaci kolejnych liczb naturalnych. Interpretacja tej skali przedstawia się następująco:

- ocena 5 - znaczenia obu fragmentów jest takie samo,
- ocena 4 lub 3 - znaczenie fragmentów jest bardzo podobne lub występuje bliska relacja
- ocena 2 lub 1 - znaczenie fragmentów jest podobne w niewielkim stopniu lub występuję jakaś relacja (mniejsza niż dla wyższych ocen i większa niż dla oceny 0),
- ocena 0 - znaczenie obu fragmentów jest kompletnie inne.

2.2. Typ dopasowania

Typ dopasowania jest rozszerzeniem oceny podobieństwa, uzupełniającym ją o warstwę wyjaśniającą daną ocenę. Występuje osiem typów dopasowanie, które interpretowane są w następujący sposób:

- EQUI - oba fragmenty mają to samo znaczenie pod względem semantycznym,
- OPPO - znaczenia fragmentów są sobie przeciwstawne,
- SPE1 - oba fragmenty mają podobne znaczenie, ale fragment w zdaniu pierwszym jest bardziej szczegółowy niż fragment w zdaniu drugim,
- SPE2 - podobnie do SPE1, przy czy czym to fragment drugi zawiera więcej szczegółów,

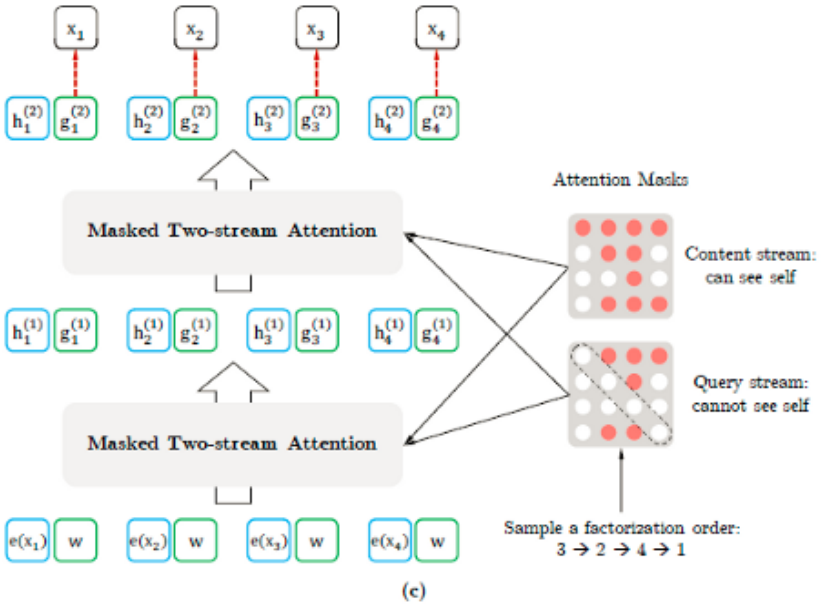
- SIMI - oba fragmenty mają podobne znaczenie i dzielą podobne atrybuty, przy czym nie można ich przydzielić do żadnej z powyższych relacji (EQUI, OPPO, SPE1, czy SPE2),
- REL - podobnie do SIMI, przy czym fragmenty nie dzielą wspólnych atrybutów. Istnieje jednak pewna bliska relacja między nimi,
- NOALI - gdy dla fragmentu z jednego ze zdań nie istnieje odpowiadający mu fragment w drugim zdaniu,
- ALIC - typ podobny do NOALI, jednak w tym przypadku brak dopasowania spowodowany jest ograniczeniem w przypisywaniu fragmentów 1:1. Bez tego ograniczenia istnieje możliwość przyporządkowania danego fragmentu.

Dodatkowo istnieją następujące reguły dotyczące powiązania typu dopasowania i oceny podobieństwa:

- Ocena podobieństwa powinna wynosić 0 tylko i wyłącznie dla typów NOALI oraz ALIC,
- Ocena podobieństwa powinna wynosić 5 tylko i wyłącznie dla typu EQUI.

3. XLNet

XLNet [2] jest to sieć neuronowa oparta o architekturę Transformer-XL. Jest to model języka AR (ang. autoregressive), który wykorzystuje operację permutacji na tokenach wejściowej sekwencji, dzięki czemu jest w stanie przechwycić kontekst dwukierunkowy sekwencji. Aby osiągnąć wspomnianą permutację bez zgubienia infromacji o prawdziwej kolejności tokenów, model ten używa kodowania pozycyjnego i specjalnej maski uwag używanej w Transformersach (patrz Rysunek 3.1).



Rysunek 3.1. Uczenie permutacjami z wykorzystaniem dwóch potoków mechanizmów uwagi. [2]

Podczas przetwarzania jednej sekwencji, XLNet bierze wejściowe tokeny i generuje z nich permutacje. Następnie na podstawie tokenów znajdujacych się po przed rozpatrywanym wyznacza kontekst. Dzięki temu, że rozpatruje on wiele permutacji to, mimo że w trakcie przetwarzania jednej z nich bierze tylko tokeny z lewej strony, i tak uzyska kontekst dwukierunkowy.

3.1. XLNet vs BERT

XLNet powstał na podstawie i w odpowiedzi na pewne bolączki modelu BERT. BERT jest modelem typu AE (ang. autoencoding), co oznacza, że odtwarza on prawdziwe dane ze „zniszczonych". Zniszczenie w tym wypadku oznacza, że niektóre tokeny zastępowane są symbolem [MASK]. Zaletą tego rozwiązania jest to, że jest ono w stanie widzieć kontekst w obu kierunkach [3]. Niestety to podejście ma również swoje wady. Zastosowanie symbolu [MASK] sprawia, że pewne słowa mogą nie pojawić się w naszym słowniku. Ponad to „zamaskowane" symbole są wyznaczane niezależnie od siebie. Tego problemu nie posiada XLNet, który rozpatruje kolejne słowa na podstawie wszystkich poprzednich, co dodatkowo sprawia, że XLNet uczy się większej liczby zależności. Dzięki tym różnicom XLNet-owi udało się osiągnąć lepsze wyniki w wielu zadaniach NLP.

4. Opis rozwiązania

Jako baza rozwiązania posłużyła nam oryginalna implementacja XLNet[4], która to dostarcza pomocnicze klasy i interfejsy mające na celu rozwiązywanie wiele rodzajów zadań z dziedziny NLP. Ów klasy, nazwane przez autorów procesorami, implementują rozwiązania zadań z klas takich jak klasyfikacja czy regresja. Przykładowo, istnieje procesor o nazwie *StsbProcessor*, który był użyty do przetestowania (po wcześniejszym douczeniu) modelu XLNet na zadaniu określania podobieństwa semantycznego (regresja) z benchmarku GLUE[5]. W celu rozwiązania zadania postawionego zadania została stworzona nowa klasa/procesor *IstsProcessor* bazująca m.in. na *StsbProcessor*, jednakże zaadaptowanej do zadania iSTS. Adaptacja ta polegała na implementacji klasyfikacji w miejsce regresji przy wykorzystaniu klas pomocniczych zapewniających kompatybilność procesorów z rdzeniem modelu XLNet. Następnie powstała klasa została zaadoptowań do skrótów umożliwiających parametryczne uruchomienie implementacji XLNet co umożliwiło douczenie pre-trenowanego modelu na na danych wejściowych. Szczegółowy opis implementacji znajduje się w rozdziale 5.

4.1. Pre-trenowany model

Do realizacji projektu wybrany został podstawowy model XLNet-a, kóry składa się z 12 warstw, 768 ukrytych warstw i i 12 głów atencji. Można również użyć (przedstawiono to w notatniku prezentującym rozwiązanie) modelu, który posiada 24 warstwy, 1024 ukryte warstwy

i 16 głów atencji, jednakże ze względu na ograniczenia zasobów i danych do fine-tuningu wybrany został model podstawowy.

4.2. Reprezentacja wyniku

Standardowe podejście realizacji zadania iSTS zakłada reprezentację wyniku predykcji podobieństwa sematycznego dwóch zdań w postaci dwóch wartości, oceny podobieństwa i typu dopasowania. W celu adaptacji tego zadania połączyliśmy te wartości w jedną poprzez wyznaczenie wszystkich sensownych permutacji. Przy czym określając sensowność bazowaliśmy na ograniczeniach wynikających z charakterystyki zadania iSTS. Dzięki czemu problem sprowadził się do przypisania dwóch sekwencji do jednej klasy reprezentowanej przez parę ocena-typ, która w żadnym stopniu nie traci na interpretowalności wyniku.

4.3. Dane uczące i testowe

Jako dane uczące wykorzystane zostały zbiory udostępnione przez organizatorów konkursu SemEval:

- „Images Captions” - zbiór podpisów obrazów (750 par zdań trenujących 375 par zdań testowych);
- „News Headlines” - nagłówki informacji prasowych (750 par zdań trenujących 375 par zdań testowych);
- „Answer-Students” - odpowiedzi uczniów, wydobyte z interakcji z tutorialowym systemem dialogowym (333 par zdań trenujących 334 par zdań testowych);

Do uczenia wykorzystane zostały reprezentacje par w formacie "word alignment", który zawiera wzorcowy podział zdań na fragmenty oraz dopasowania fragmentów ze zdania pierwszego do fragmentów zdania drugiego. Każdemu dopasowaniu jest przypisana ocena i typ podobieństwa.

Oczywiście dane te wymagały wstępnego przetworzenia, na które składało się:

- wyodrębnienie sekwencji z reprezentacji wzorcowego podziału tj. wyciągnięcie porównywanych sekwencji wraz z ocenami z fragmentów zagregowanych dla całych zdań,
- Usunięcie nieprawidłowych par sekwencji:
  — zawierających tylko jedną sekwencje w parze tj. brak dopasowania - wynika to z charakterystyki zadania, które realizowaliśmy,
  — usunięcie nieprawidłowych wartości dopasowania tj. niespełniających warunków zadania.

4.4. Hiper-parametry

Parametry użyte podczas procesu douczania (ang. fine-tuningu) zostały zapożyczone z oficjalnego artykułu prezentującego XLNet-a[2] oraz dobrane do dostępnego GPU na Colab-ie. Wartości hiper-parametrów wykorzystane podczas fine-tuningu to:

- *learning rate* dla optymalizatora Adam - 2e-5,
- *batch size* dla trenowania - 32,
- maksymalna długość sekwencji - 128,
- liczba kroków trenujących - 1500,
- liczba kroków rozgrzewających (mające uniknąć zapominaniu) - 250.

5. Implementacja

Zaimplementowanie rozwiązanie można podzielić na cztery główne segmenty:

- pre-procesor - mający na celu wstępne przetworzenie danych,
- procesor - elementy implementujący logikę zadania iSTS do bazowej implementacji XLNet,
- metody odpowiadające za wyznaczanie metryk oceny,
- plik Colaboratory (notatnik w Google Colab) - odpowiadający za formalizację sekwencji korków wywoływania odpowiednich metod począwszy od preprocessingu kończąc na uzyskaniu metryk.

Oczywiście każdy z wyżej wymienionych segmentów zawiera w sobie również integrację z istniejącym rozwiązaniem[4] oraz metody pomocnicze, które nie zostaną przedstawione w szczegółowym przedstawieniu poszczególnych segmentów (znajdującym się poniżej) lub zostaną tylko wspomniane.

Repozytorium zawierające rozwiązanie końcowe dostępne jest jako publiczne na serwisie GitHub[6] jako fork oryginalnego rozwiązania.

5.1. Pre-procesor

Za preporcessing odpowiada metoda *preprocess_data*, której implementacja została przedstawiona poniżej. Metoda ta wykonuje czynności opisane w podrozdziale 4.2. W ogólności działanie tej metody sprowadza się do:

1. wczytania pliku zawierającego dopasowania w postaci złotego podziału,
2. odfiltrowania par sekwencji wraz z oceną i typem dopasowania,
3. przetworzeniu danych do docelowej postaci tj. pary sekwencji wraz z połączoną oceną i typem dopasowania w postaci <*typ_dopasowania*>-<*ocena*>,
4. zapisie wynikowych danych w pliku o wskazanej ścieżce.

```
def preprocess_data(input, output):
    assert os.path.exists(input), "Input file: {} does not
     ↪  exists".format(input)

    file = open(input, 'r+')
    data = file.read()
    results = []
```

```
    for alignments in
     ↪ re.finditer("(<alignment>(.|\\n|\\r\\n)*?<\\/alignment>)",
     ↪ data):
        alignments = re.split('\\n', alignments.group(0))
        for alignment in alignments[1:-1]:
            elements = alignment.split("//")
            if '-not aligned-' in [x.strip() for x in
             ↪ elements[3].split("<==>")]:
                continue
            type = elements[1].replace(" ", "").split("_")[0]
            score = elements[2].replace(" ", "")
            score = '0' if elements[2].replace(" ", "") == 'NIL' else
             ↪ score
            texts = elements[3].split("<==>")
            textA = texts[0].strip()
            textB = texts[1].strip()
            tokens = elements[0].split("<==>")
            num_tokens = min(len(list(map(int, tokens[0].split()))),
             ↪ len(list(map(int, tokens[1].split()))))
            res = [type+"-"+score, textA, textB, num_tokens]
            if res[0] not in RESULTS:
                print("WARRNING: Wrong input data: {}".format(res[0]))
                continue
            results.append(res)
    with open(output, 'w', newline='') as myfile:
        wr = csv.writer(myfile, delimiter='\t', lineterminator='\n')
        wr.writerows(results)
```

5.2. Procesor

Procesor jest głównym elementem pozwalającym na adaptacje XLNet-a do różnorakich zadań z dziedziny NLP. A jego zadaniem (pośrednim) jest adaptacja warstwy wyjściowej sieci neuronowej oraz dostarczanie danych zarówno testowych jaki i trenujących. W wyniku implementacji powstała klasa *IStsProcessor* (implementacja przedstawiona poniżej), która dziedziczy po klasie *GLUEProcessor*. *GLUEProcessor* dostarcza podstawowych metod zwracających dane testowe i trenujące poprzez wywołanie metody *_create_examples* dla odpowiednich plików. Ścieżka do katalogów, w których znajdują się pliki przekazywana jest przy pomocy flag podczas wywołania, natomiast konwencją jest aby same pliki nazywały się *test.tsv* (dane testowe) oraz *train.tsv* (dane trenujące).

```
class IStsProcessor(GLUEProcessor):
    def __init__(self):
        super(IStsProcessor, self).__init__()
        self.label_column = 0
        self.text_a_column = 1
        self.text_b_column = 2
        self.contains_header = False
        self.test_contains_header = False

    def get_labels(self):
        return ["EQUI-5", "OPPO-1", "OPPO-2", "OPPO-3", "OPPO-4",
                "SPE1-1", "SPE1-2", "SPE1-3", "SPE1-4",
                "SPE2-1", "SPE2-2", "SPE2-3", "SPE2-4",
                "SIMI-1", "SIMI-2", "SIMI-3", "SIMI-4",
                "REL-1", "REL-2", "REL-3", "REL-4"]

    def _create_examples(self, lines, set_type):
        examples = []
        for (i, line) in enumerate(lines):
            if i == 0 and self.contains_header and set_type != "test":
                continue
            if i == 0 and self.test_contains_header and set_type ==
             ↪  "test":
                continue
            guid = "%s-%s" % (set_type, i)

            if len(line) <= self.text_a_column:
                tf.logging.warning('Incomplete line, ignored.')
                continue
            text_a = line[self.text_a_column]

            if len(line) <= self.text_b_column:
                tf.logging.warning('Incomplete line, ignored.')
                continue
            text_b = line[self.text_b_column]

            label = line[self.label_column]
            examples.append(
                InputExample(guid=guid, text_a=text_a, text_b=text_b,
                 ↪  label=label))
```

```
        return examples
```

5.3. Metryki oceny rozwiązania

Szczegółowy opis metryk przedstawiony został w pracy Pani Eweliny Grudzień[1]. Za wyznaczanie metryk w naszym rozwiązaniu odpowiadają klasy *F1Metrics* oraz *PearsonMetrics*, które implementują odpowiadające ich nazwom metryki w sposób zgodny z pracą magisterską i konkursem SemEval. Poniżej przedstawiono (w skróconej formie) implementacje ów klas.

```
class F1Metrics:
    ...

    def f1_type_match(self):
        ...
        return self.count_f1(overlap, overlap)

    def f1_score_match(self):
        ...
        return self.count_f1(overlap, overlap)

    def f1_all_match(self):
        ...
        return self.count_f1(overlap, overlap)

    def is_special_case1(self, predicted_type, predicted_score,
     ↪ target_type, target_score):
        ...
        return False

    def is_special_case2(self, predicted_type, predicted_score,
     ↪ target_type, target_score):
        ...
        return False

    def count_f1(self, overlap_gs, overlap_sys):
        ...
        return 2 * precision * recall/ (precision + recall)
```

```
class PearsonMetrics:
    ...

    def count_person_for_type(self):
        ...
        return
         ↪  self.count_pearson(self.get_types_as_values(self._predicted_types),
         ↪  self.get_types_as_values(self._target_types),
                                    mean_p, mean_t)

    def count_mean_for_type(self, types):
        return self.count_mean(self.get_types_as_values(types))

    def count_pearson_for_score(self):
        ...
        return self.count_pearson(self._predicted_scores,
         ↪  self._target_scores, mean_p, mean_t)

    def count_mean_score(self, scores):
        return self.count_mean(scores)

    def count_pearson(self, predicted, target, mean_p, mean_t):
        ...
        return up / (sum_p * sum_t)

    def get_types_as_values(self, types):
        return list(map(TYPE_VALUE_MAP.get, types))

    def count_mean(self, values):
        sum_val = sum(values)
        return sum_val / len(values)
```

5.4. Notatnik Colaboratory

Notatnik z rozwiązaniem znajduje się w repozytorium projektu oraz jest dostępny pod linkiem [7]. Zawiera on wszystkie niezbędne kroki potrzebne do zreprodukowania wyników (z pewnym przybliżeniem) przedstawionych w dokumentacji.

Notatnik składa się z następujących sekcji:

1. Setup - pobranie zależności, repozytorium itp.,
2. Definiowanie zmiennych

3. Wstępne przetwarzanie danych
4. Trenowanie modelu
5. Predykcja klas przy użyciu wytrenowanego modelu
6. Wyznaczanie metryk dla zadania iSTS (określonych w specyfikacji konkursu SemVal)
7. Push-owanie (ewentulanych) zmian/wyników na repozytorium - sekcja deweloperska
8. Kopiowanie plików z/do Google Drive-a - sekcja deweloperska

6. Instrukcja obsługi

W celu uruchomienia zaimplementowanego rozwiązania należy otworzyć dostarczony link kierujący do notatnika w Google Colaboratory[7] i postępować zgodnie z przedstawionymi tam krokami. Sprowadza się to do uruchomienia kolejnych kroków, chyba, że opis kroku wskazuje inaczej. W przypadku chęci uruchomienia rozwiązania na TPU, lokalnych maszynach lub innych problemów (np. coś zmieniło się z konfiguracji Google Colaboratory) odsyłamy do oficjalnego repozytorium XLNet[4] (instrukcja i sekcja Issues).

7. Test

7.1. Miary oceny

Do oceny naszego modelu wykorzystane zostały miary F1 i współczynnik korelacji Pearsona, które zostały obliczone na podstawie re-implementacji skryptu dostarczonego przez autorów konkursu oraz Panią Ewelinę Grudzień w jej pracy magisterskiej.

Miary F1 obliczane są w trzech wariantach:

1. dopasowany identyfikator powinien być zgodny, ale oceny w skali liczbowej są ignorowane,
2. dopasowany identyfikator jest ignorowany, ale każde dopasowanie jest karane, gdy ocena w skali liczbowej jest niezgodna,
3. dopasowany identyfikator powinien być zgodny oraz każde dopasowanie jest karane, gdy ocena w skali liczbowej jest niezgodna.

Współczynnik korelacji Pearsona obliczany są w dwóch wariantach:

1. dopasowany identyfikator powinien być zgodny, ale oceny w skali liczbowej są ignorowane,
2. dopasowany identyfikator jest ignorowany, ale każde dopasowanie jest karane, gdy ocena w skali liczbowej jest niezgodna.

7.1.1. Przypadki specjalne

Przy obliczaniu miary F1 zostały wzięte pod uwagę następujące przypadki specjalne:

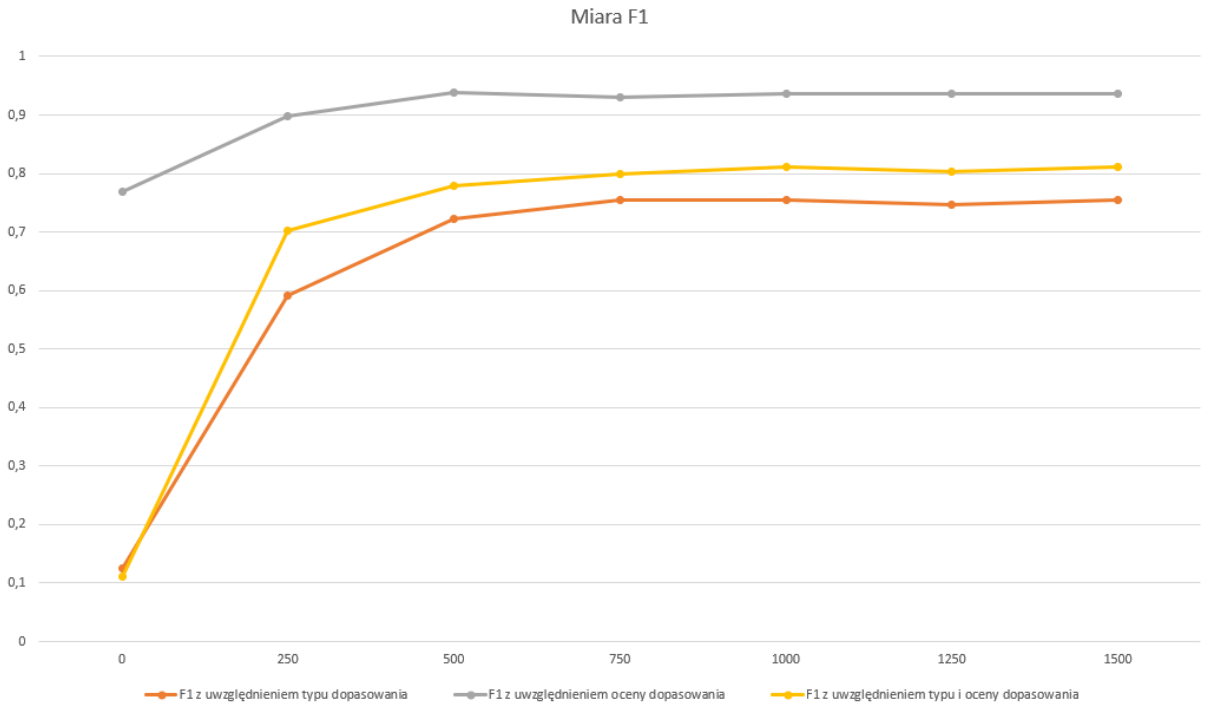
- kara za rozbieżność w typach: SPE1, SPE2, REL, SIMI jest pomijana, jeśli obie oceny należą do przedziału (0-2];

- kara za rozbieżność w typach EQUI oraz SIMI/SPE jest pomijana, jeśli ocena wynosi 4;

7.2. Wyniki

Testy zostały przeprowadzone na danych wymienionych w 4.3. Dla każdego zbioru danych sprawdzaliśmy wyniki co 250 kroków uczenia (od 0 do 1500, gdzie 0 - model XLNet bez fine-tuning'u).

7.2.1. Zbiór *Images*



Rysunek 7.1. Metryka F1 dla zbioru *Images*.

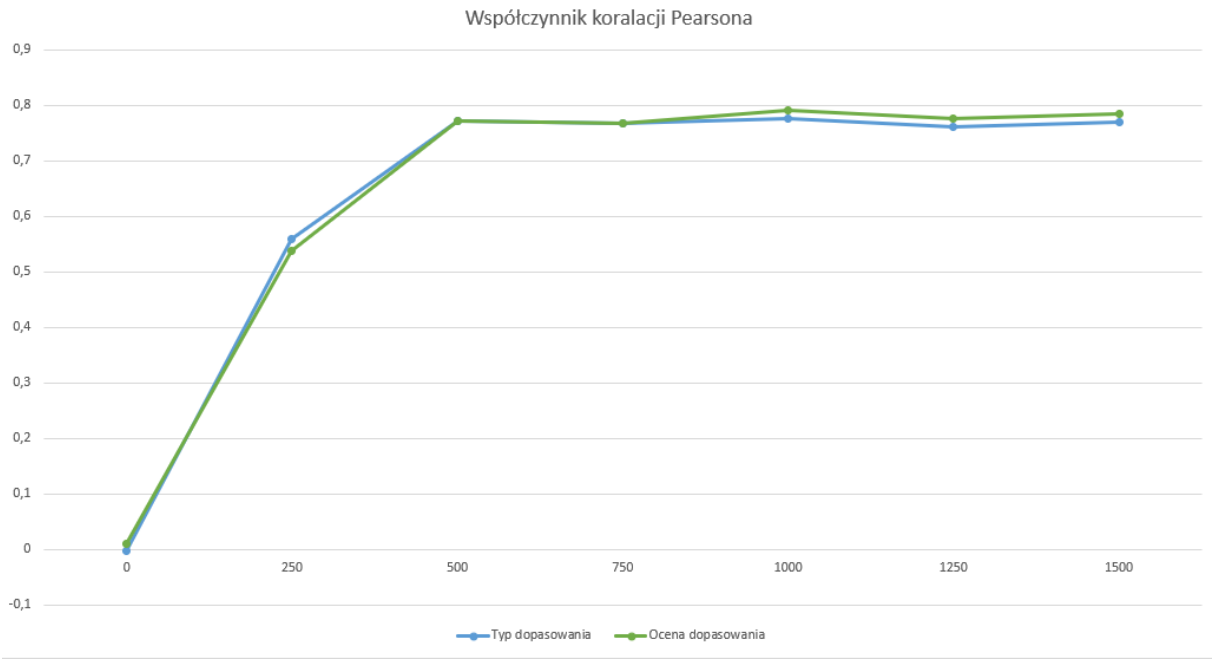
Dla zbioru *Images* najlepsze osiągnięte wyniki po 1500 krokach to:

- F1 z uwzględnieniem typu: **0,754**
- F1 z uwzględnieniem oceny: **0,936**
- F1 z uwzględnieniem typu i oceny: **0,812**
- Współczynnik korelacji Pearson'a dla typu dopasowania: **0,771**
- Współczynnik korelacji Pearson'a dla oceny dopasowania: **0,786**

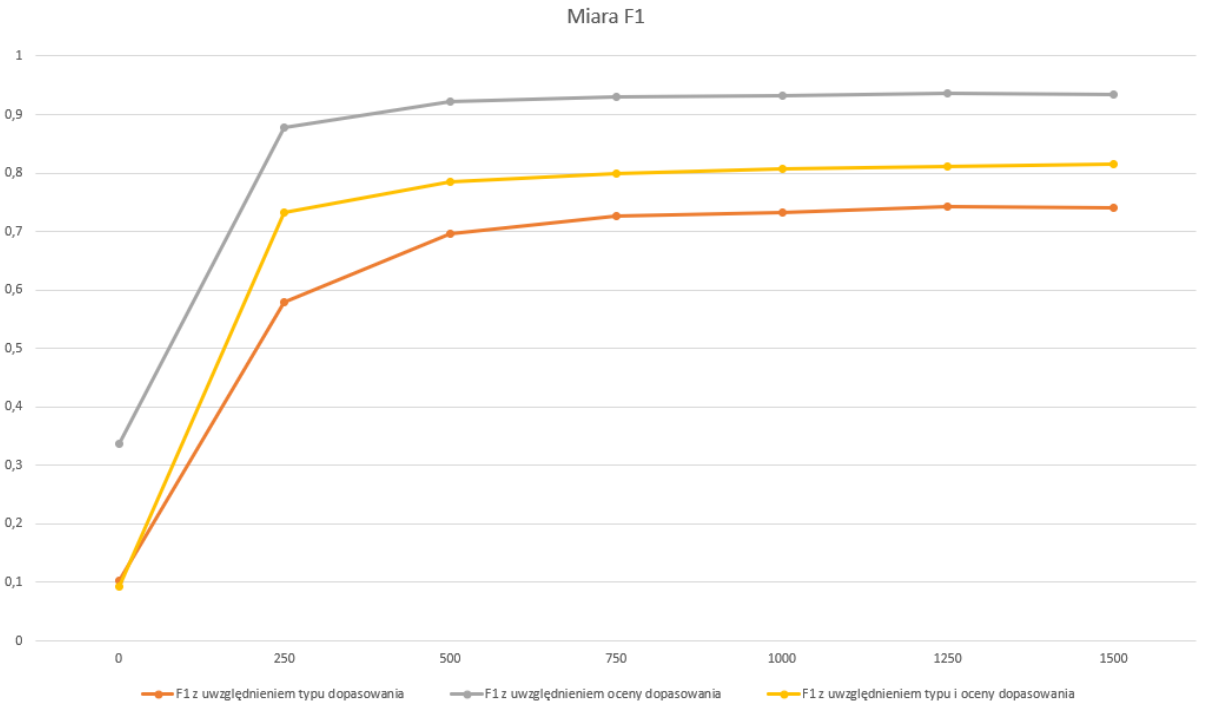
7.2.2. Zbiór *Headlines*

Dla zbioru *Headlines* najlepsze osiągnięte wyniki po 1250 krokach to:

- F1 z uwzględnieniem typu: **0,743**
- F1 z uwzględnieniem oceny: **0,936**
- F1 z uwzględnieniem typu i oceny: **0,812**
- Współczynnik korelacji Pearson'a dla typu dopasowania: **0,728**
- Współczynnik korelacji Pearson'a dla oceny dopasowania: **0,770**



Rysunek 7.2. Współczynnik Pearsona dla zbioru *Images*.

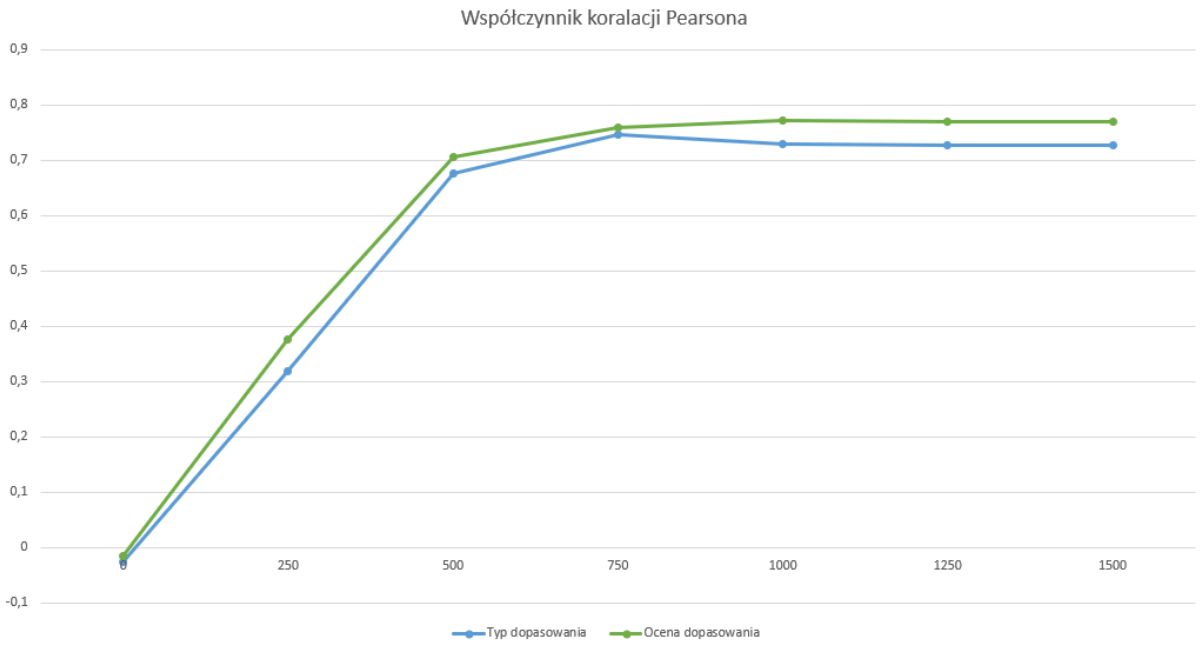


Rysunek 7.3. Metryka F1 dla zbioru *Headlines*.

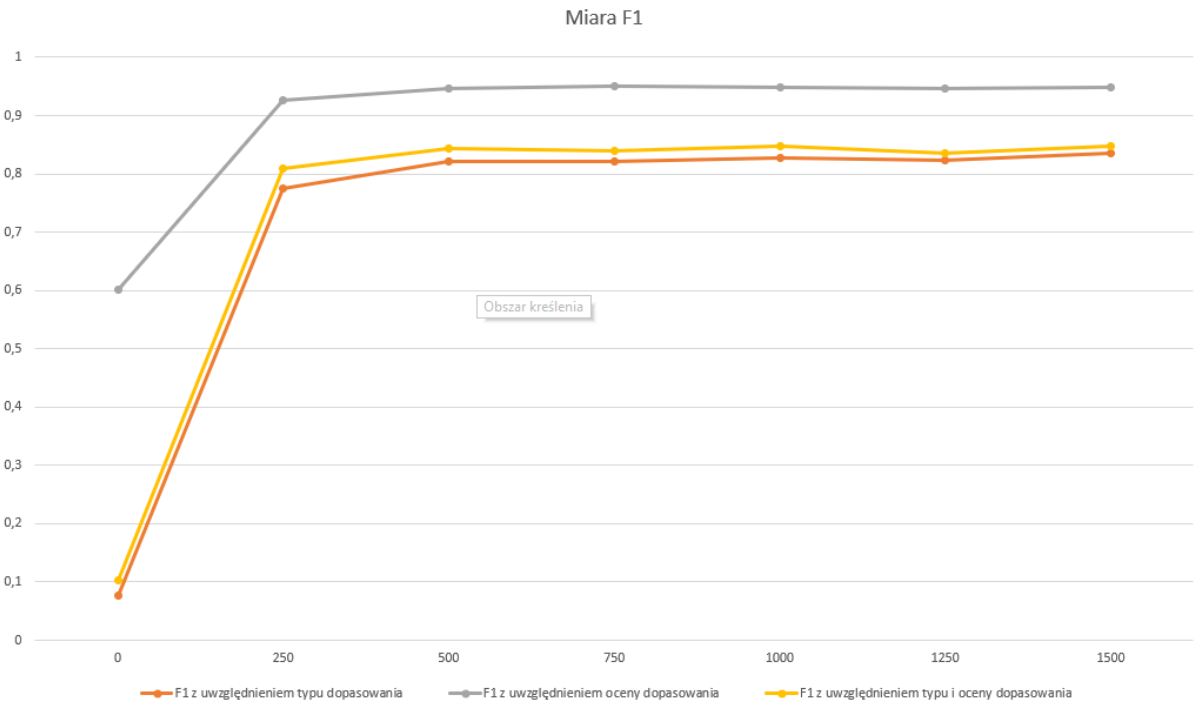
7.2.3. Zbiór *Answers Students*

Dla zbioru *Answers Students* najlepsze osiągnięte wyniki po 1500 krokach to:

- F1 z uwzględnieniem typu: **0,835**
- F1 z uwzględnieniem oceny: **0,949**
- F1 z uwzględnieniem typu i oceny: **0,847**

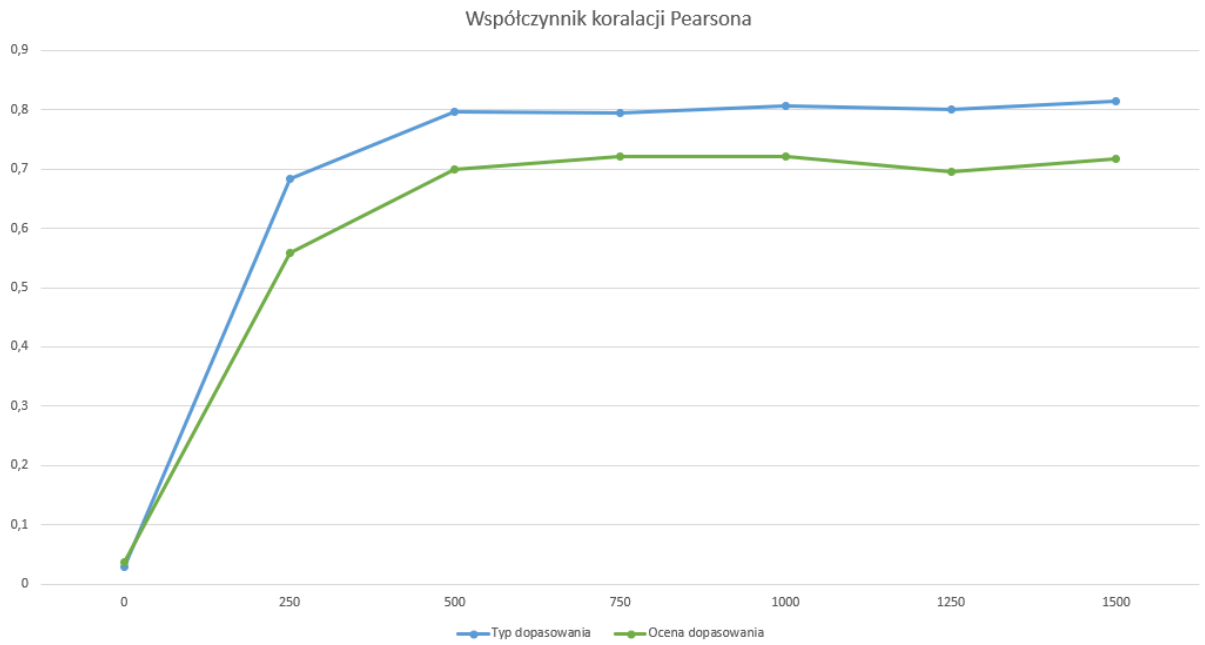


Rysunek 7.4. Współczynnik Pearsona dla zbioru *Headlines*.



Rysunek 7.5. Metryka F1 dla zbioru *Answers Students*

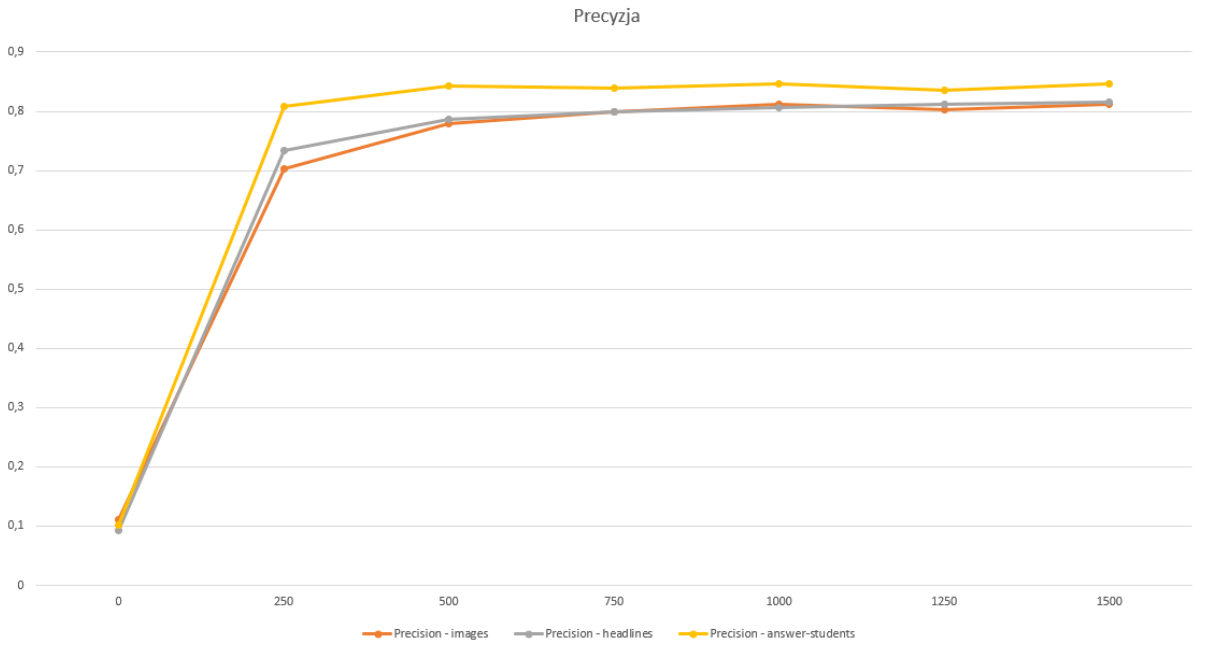
- Współczynnik korelacji Pearson'a dla typu dopasowania: **0,814**
- Współczynnik korelacji Pearson'a dla oceny dopasowania: **0,717**



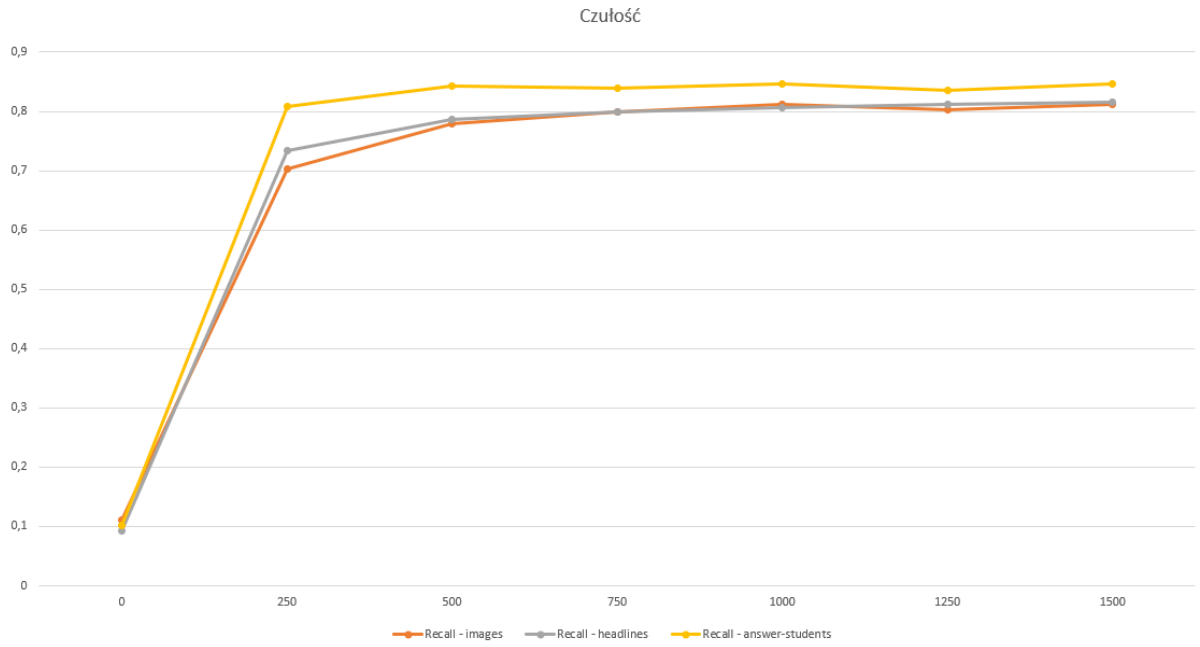
Rysunek 7.6. Współczynnik Pearsona dla zbioru *Answers Students*.

7.2.4. Precyzja i czułość

Poniżej zostały zestawione zmiany miary precyzji (ang. precision) i czułości (ang. recall) dla zbiorów testowych. Jak widać również te miary bardzo szybko osiągają swoje maksimum.



Rysunek 7.7. Miara precyzji dla wszystkich zbiorów.



Rysunek 7.8. Miara czułości dla wszystkich zbiorów.

7.3. Porównanie z wynikami innych sieci

Dla sieci opisanych w [1] najlepsze wyniki osiągnęła sieć typu VANILLA MULTI_H DEPENDENCY TREE.

- F1 z uwzględnieniem oceny: 0,917
- F1 z uwzględnieniem typu: 0,719
- F1 z uwzględnieniem typu i oceny: 0,745

Jak widać model XLNet po finetuningu potrafi osiągnąć jeszcze lepsze wyniki od tych. Wynikać to może z tego, że jest to model, który został już nauczony zależności pomiędzy słowami (potrafi rozumieć sens zdań) i wystarczy tylko wskazać mu w procesie finetuningu w jaki sposób ma oceniać zdania.

8. Wnioski i obserwacje

Wynik otrzymane przy pomocy rozwiązania opartego o XLNet są wyraźnie lepsze niż wynik otrzymane w pracy magisterskiej Pani Eweliny Grudzień. Jednakże, nie powinno to dziwić, ponieważ rozwiązanie opisane w pracy magisterskiej bazuje na modelach, które nie były wcześniej trenowane i miały one do dyspozycji jedynie zbiory danych dostarczone przez twórców konkursu, które są dosyć mocno okrojone. W przypadku korzystania z pre-trenowanego modelu, którym jest XLNet, sytuacja ma się kompletnie inaczej, nie bez powodu takie rozwiązanie nosi nazwę modelu językowego. XLNet był pre-trenowany na olbrzymich korpusach danych (ok. 33 miliardy *subowrd pieces*) przez ok. 5.5 dnia (dane dla XLNet-Large) a co za tym

idzie miał znaczną przewagę już na starcie. Dlatego też zestawieni takiego modelu z modelem nie pre-trenowanym daje znacznie lepsze rezultaty.

Można również zaobserwować, że uzyskane wyniki nie różnią się zbytnio pomiędzy zbiorami danych. Może to wynikać z faktu, że model XLNet potrafi rozpoznać znaczenie zdań (sekwencji tokenów), a proces finetuningu jedynie uczy go wskazywać odpowiednią klasę oceny.

Dla wszystkich zbiorów można zaobserwować szybki wzrost jakości w pierwszych 500 krokach, później wyniki utrzymują się na podobnym poziomie. Ponownie, może to wynikać z faktu, że mamy do czynienia z pre-trenowanym modelem, który doskonale zna rozpoznaje zależności semantyczne pomiędzy zdaniami.

Bibliografia

[1] E. Grudzień, „Rozpoznawanie podobieństwa semantycznego z wykorzystaniem architektur uczenia głębokiego”, prac. mag., Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska, 2018.

[2] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov i Q. V. Le, *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, 2020. arXiv: 1906.08237 [cs.CL].

[3] X. Liang, „What is XLNet and why it outperforms BERT”, 2019. adr.: https://towardsdatascience.com/what-is-xlnet-and-why-it-outperforms-bert-8d8fce710335.

[4] *XLNet*, Dostęp zdalny (08.04.2020): https://github.com/zihangdai/xlnet.

[5] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy i S. R. Bowman, *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*, 2019. arXiv: 1804.07461 [cs.CL].

[6] *XLNet - iSTS*, Dostęp zdalny (01.06.2020): https://github.com/agrudkow/xlnet.

[7] *XLNet - iSTS - Google Colab*, Dostęp zdalny (01.06.2020): https://colab.research.google.com/github/agrudkow/xlnet/blob/master/notebooks/colab_imdb_gpu.ipynb.